Amendment to the Specification

Please amend the specification as follows.

Please replace the paragraph on page 2, under the heading "CROSS REFERENCE TO RELATED APPLICATIONS", with the following corrected paragraph:

This invention ~~relates to~~ claims the benefit of Provisional Application 60/451,490.

On page 5, under the heading "BRIEF DESCRIPTION OF THE DRAWING", please insert the following after the first paragraph:

Fig. 2 is a diagram of a system initialization process of the system of Fig. 1;

Fig. 3 is a diagram of a register add process of the system of Fig. 1;

Fig. 4 is a diagram of a window add process of the system of Fig. 1;

Fig. 5 is a diagram of a populate window with historical data process of the system of Fig. 1;

Fig. 6 is a diagram of a persistent query add process of the system of Fig. 1; and

Fig. 7 is a diagram of a point add process of the system of Fig. 1.

On pages 6 et seq., please replace the paragraphs beginning under the heading "Parts" with the following corrected paragraphs:

The time series monitoring system can be implemented using a software class called an engine. The engine 100 comprises a filter 102, a sequencer 104, a set of registers 106, and a set of persistent queries 108. The engine 100 also coordinates methods to insert data 112 into the system, execute persistent queries 114, and to dynamically manage registers 106, windows 110, and persistent queries 108.

Each time series monitored by the system corresponds to a register 106. Each register 106 has a ring buffer 116 to store time series values. Each register 106 may contain one or more windows 110. Each window 110 is associated with a subset of the time series. Each window 110 maintains statistics for the associated subset of the time series.

Each persistent query 108 comprises an event condition 120 and a payload specification 122. The event condition 120 is a Boolean function that may depend on statistics from multiple registers 106. The payload specification 122 expresses which data are to be output by the engine 100 when the event condition 120 is evaluated as true. For each persistent query 108, there is a set of associated registers 106 called trigger registers. The engine 100 evaluates the event condition 108 in response to receiving a value from any time series corresponding to a trigger register 106.

Response to an Input from a Time Series

The engine 100 receives as input a stream of labeled data 112, in which each value is associated with a label indicating the time series to which the value belongs. For each input value, the filter 102 uses the label to determine whether the value belongs to a time series monitored by the system. If not, then the value is discarded. If so, then the engine 100 forms a point that comprises the value, a timestamp, and a unique id number supplied by a sequencer 104. Then the engine 100 sends the point to the register 106 that corresponds to the time series to which the value belongs.

The register 106 installs the point in the buffer 116 of the register 106. Next, each window 110 in the register 106 updates the statistics for the window 110 using online computation, accounting for the new point and for any points that have expired from the window 110 since the previous

statistics computation. Then the register 106 adjusts the buffer 116, discarding points that have expired from all windows 110, growing the buffer 116 if needed to accommodate new points, and shrinking the buffer 116 if needed to accommodate storage needs of other buffers 116.

Then the engine 100 executes each persistent query 108 for which the register 106 is a trigger. Each persistent query 108 comprises an event condition 120 and a payload specification 122. The engine 100 evaluates the event condition 120. If the event condition 108 is true, then the engine 100 creates a payload 124 according to the payload specification 122 and delivers the payload 124 as output.

Persistent Query Syntax and Parsing

A persistent query 108 may be expressed using a query string 114. The query string syntax is similar to SQL, the standard database query language. The format for a query string 114 is as follows.

SELECT <columns> FROM <trigger registers> WHERE <condition>

Columns

The columns encode the payload specification 122. Column syntax is as follows.

<columns> = REISTER.WINDOW.NAME [[, REGISTER.WINDOW.NAME [,...]]

This is a comma-separated list of identifiers of the form REGISTER.WINDOW.NAME where NAME is the name of a statistic (like COUNT, SUM, MEAN, STDEV), one of (VALID, ALL, LAST, VAL), or a number. VALID is true when the window 110 is valid (has enough points in it or enough time in it) and false if not. ALL indicates the set of points

that entered the window 110 since the last time the event condition 108 evaluated as true. If the window 110 has received a new point since the event condition 120 last evaluated to true, the LAST indicates the newest point in the window 110, and VAL is the time series value in that point. Otherwise, LAST and CAL are both null. (The behavior for VALID AND VAL is slightly different in evaluation of event conditions 120. The differences are explained later, in the "Condition" subsection.) A number indicates the point at the position in the window 110 specified by the number, with the number 0 indicating the oldest point. For example, "WINDOW.0, WINDOW.1, WINDOW.2" indicates the oldest three points in WINDOW.

The payload 124 is delivered as a pair of lists. One list contains the column names specified in the query string 114. The other list contains the corresponding values.

Trigger Registers

The persistent query execution system evaluates the event condition 120 each time a point is added to any register 106 in the set of trigger registers 106 for the persistent query 108. The trigger registers syntax is as follows.

<trigger registers. = reg1 [[, reg2 [,...]]

This is a comma-separated list of registers 106.

Condition

The event condition 120 is encoded in the section of the query denoted by

<condition>

The syntax is that of a Java expression, extended to allow use of identifiers in the same REGISTER. WINDOW.NAME syntax as in the payload specification 122. Allowed names include statistics, VALID, and VAL. All such identifiers resolve to primitive double type values. In the expression, VALID is 1.0 if the window 110 is valid and 0.0 otherwise. The name VAL refers to the value in the last point added to the window 110.

Example

For example, consider the following query string 114.

SELECT ABC.50MEAN, DEF.5MIN.VAL, DEF.1MIN.ALL, DEF.50.STDEV FROM ABC, DEF WHERE ABC.50.MEAN>DEF.50.MEAN + DEF.50.STDEV

After adding the corresponding query 108 to an engine 100, the following occurs. After each new point is inserted into register ABC or DEF, the condition ABC.50.MEAN > DEF.50.MEAN + DEF.50.STDEV is evaluated. If the condition is true, then the payload ABC.50.MEAN, DEF.5MIN.VAL, DEF.1MIN.ALL, DEF.50.STDEV is output.

The process of converting a query string 114 to a persistent query 108 is called compilation. Two different compilation methods are dynamic compilation and runtime compilation. In dynamic compilation, a query parser 126 class takes the query string 114 as input and produces a Java program. (The query parser 126 class translates the query 114 into a standard form, ensures that the system can satisfy references to identifiers, and creates a fetcher class for each identifier.) Then a dynamic Java package uses the Java program to make a proxy class for the event condition evaluator. In runtime compilation, the java compiler javac and the disk are used to generate class bytes directly.

Processes

The following are step-by-step descriptions of some system processes 130, including details about how inter-process synchronization allows some processes to proceed concurrently. There are descriptions of processes to initialize the system 100, add a register 106, add a window 110, add a persistent query 108, and add a point.

A. Initialize the System

The system 100 is initialized by creating a new engine instance or by loading a persisted engine instance from a file, using the following steps.

1. Create a sequencer. (The sequencer 104 provides a unique number for each input value. The sequence numbers are used to ensure that each result is detected exactly once.)
2. Create a parser. (The parser 126 plays a role in converting query strings 114 to persistent queries 108.)
3. Set a compilation directory. (The compilation directory is used as a container for class files during runtime compilation of persistent queries 108.)
4. Register an event listener with the system. (As part of the event detection and payload output functions, the event listener receives notifications when the system 100 detects true event conditions and receives output payloads 124).

When de-persisting an engine 100 from a file, initialization also contains these steps:

5. De-persist and add to the engine 100 any registers 106 and any windows 100 specified by the ~~file~~ file.
6. Compile any query strings 114 specified by the file to form persistent queries 108. Add the persistent queries 108 to the engine 100.

After this process, the engine 100 is ready to undertake other methods, including adding registers 116, windows 100, persistent queries 108, and

data points. These other methods can be performed concurrently with each other after the engine 100 is instantiated.

B. Add a Register

The method to add a register 106 to the engine 100 comprises the following steps:

1. Add the register 106 to the list of engine registers.
2. Add a reference to the register 106 to a hashtable with register names as keys. This hashtable enables fast lookup of a register 106 by name.
3. If the register 106 contains any windows 110, add each window name, in the format REGISTER.WINDOW, to an engine hashtable with window names as keys. This hashtable enables fast lookup of a window 110 by register and window name.

C. Add a Window

The process to add a window 110 to a register 106 involves obtaining and releasing locks in order to ensure mutual exclusion between adding a window 110 and parts of other processes. For each register 106, there are two locks, a basic lock and a booster lock. Also, each window 110 has a lock. Obtaining a lock may involve waiting for the lock to be released by another process. Releasing a lock allows it to be obtained by another process. The process to add a window 110 to a register 106 comprises the following steps.

1. Obtain the basic register lock in order to ensure mutual exclusion with the process of adding a point to the register 106.
2. Set the number of points in the window 110 to zero, and set window endpoints to indicate that the most recent point in the register 106 is one position beyond the points in the window 110.
3. Associate the window 110 with the register 106, adding the window 110 to the list of windows 110 to be updated for each input point received by the register 106.

4. Add the window name, in the format REGISTER.WINDOW, to an engine hashtable with window names as keys. This hashtable enables fast lookup of the window 110 by register and window name.

5. Release the basic register lock.

6. To use historical data to populate the window 110, use the following iterative process.

6a. Obtain the register booster lock. Obtain the register basic lock.

6b. Get a reference to the next historical point to add to the window 110, i.e., a reference to the most recent point in the register 106 that was added before the window 110 and is not in the window 110.

6c. Obtain the window lock.

6d. Release the register basic lock. Release the register booster lock.

6e. Update the window statistics to account for the referenced point.

6f. Update the window endpoints to indicate that the referenced point is in the window 110.

6g. Release the window lock.

6h. Determine whether the process of populating the window 110 is complete as follows. For a value window 110, check whether the next historical point to be added falls before the beginning of the time interval. If the process is not complete, then go to step 6a.

D. Add a Persistent Query

The engine 100 has a query lock, which is used to ensure mutual exclusion between the process of adding a persistent query 108 and the process of executing a persistent query 108. The process to add a persistent query 108, specified by a query string 114, to the engine 100 comprises the following steps.

1. Obtain the query lock.

2. Parse the query to obtain strings corresponding to the payload specification 122, trigger registers 106, and event condition 120.

3. Parse the payload specification, trigger registers, and event condition strings to determine all references to registers 106, windows 110, and names. Check that the referenced entities exist. If so, then proceed. If not, then release the query lock and exit this process.
4. Compile the query string 114 to form a persistent query 108, using runtime or dynamic compilation as described previously.
5. Register the persistent query 108 with any ~~with any~~ trigger registers 106 of the persistent query 108.
6. Release the query lock.

E. Add a Point

Each input value 112 is accompanied by a label that indicates the time series to which the value belongs. The process by which the engine 100 handles an input time series value comprises the following steps.
1. Use the filter 102 to determine whether the label refers to a time series that corresponds to a register 106 in the engine 100. If ~~not~~ not, then exit this process.
2. Use the sequencer 104 to generate a unique sequence number for the value. Form a point that comprises the value and the unique sequence number.
3. Obtain the register basic lock. This ensures that no window 110 is added to the register 106 and that no other point is added to the register 106 while the point is being added to the register 106.
4. Check whether the size of the buffer 116 in the register 106 is large enough to include the point in addition to any previous points in the buffer 116. If the buffer 116 is not large enough; then boost the buffer size as follows.
4a. Obtain the booster lock on the register 106. This ensures mutual exclusion with part of the process to populate a new window 110 with historical points.
4b. Allocate storage to increase buffer size.

4c. Copy existing points into any corresponding positions of newly allocated buffer storage.

4d. Update and window references to buffer locations.

4e. Release the register booster lock.

(A similar process to steps 4a to 4e is used to shrink the buffer size when little of the buffer 116 is being used after data expiry or window deletion.)

5. For each window 110 of the register 106, (a) obtain the window lock; (b) update the window 110 to account for the new point; (c) determine which points (if any) have expired from the window 110, and update statistics to account for expiry; (d) release the window lock.

6. Discard points from the register buffer that will have no further effect on statistics or values for any window 110.

7. For each persistent query 108 for which the register 106 is a trigger, do the ~~following.~~ following: Obtain the query lock. Evaluate the event condition 120. If the event condition 120 is true, then fetch and output the payload 124 as specified by the payload specification 122. Then release the query lock.